# Fostering software quality assessment

Brandtner, Martin

Abstract: Software quality assessment shall monitor and guide the evolution of a system based on quality measurements. This continuous process should ideally involve multiple stakeholders and provide adequate information for each of them to use. We want to support an effective selection of quality measurements based on the type of software and individual information needs of the involved stakeholders. We propose an approach that brings together quality measurements and individual information needs for a context-sensitive tailoring of information related to a software quality assessment. We address the following research question: How can we better support different stakeholders in the quality assessment of a software system? For that we will devise theories, models, and prototypes to capture their individual information needs, tailor information from software repositories to these needs, and enable a contextual analysis of the quality aspects. Such a context-sensitive tailoring will provide a effective and individual view on the latest development trends in a project.

DOI: https://doi.org/10.1109/ICSE.2013.6606725

# Fostering Software Quality Assessment

Martin Brandtner
Department of Informatics, University of Zurich, 8050 Zurich, Switzerland
Web: http://seal.ifi.uzh.ch/brandtner, E-Mail: brandtner@ifi.uzh.ch

***Abstract*—Software quality assessment shall monitor and guide the evolution of a system based on quality measurements. This continuous process should ideally involve multiple stakeholders and provide adequate information for each of them to use. We want to support an effective selection of quality measurements based on the type of software and individual information needs of the involved stakeholders. We propose an approach that brings together quality measurements and individual information needs for a context-sensitive tailoring of information related to a software quality assessment. We address the following research question: How can we better support different stakeholders in the quality assessment of a software system? For that we will devise theories, models, and prototypes to capture their individual information needs, tailor information from software repositories to these needs, and enable a contextual analysis of the quality aspects. Such a context-sensitive tailoring will provide a effective and individual view on the latest development trends in a project. We outline the milestones as well as evaluation approaches in this paper.**

## I. Introduction

Crosby argued that *it is not quality that is expensive, but rather the lack of it* [3]. Quality of software can be assessed in different ways and a software quality assessment (SQA) should take a wide range of different aspects into account. Such a large set of quality aspects cannot be assessed by a single kind of stakeholders, for example, software testers. Typical quality aspects are different properties of source code such as cohesion or coupling [2], test-related properties such as coverage, or architecture-related properties, such as communication directives or data access constraints. A coverage of all these aspects is only possible when SQA involves different stakeholder groups.

As a consequence, software quality is more than just source code metrics that can be rather easily computed for a software project. Another essential dimension are the specific information needs of the different stakeholders, who are involved in SQA. The raw data for an SQA that is available to them is stored in various data formats and repositories such as version control systems (VCS), issue trackers, or wikis, etc.

One essential and time-consuming task for a stakeholder during a typical SQA is to get the adequate information [9], [11] out of a bulk of data. The value of information varies per stakeholder and her actual role in the project and experience. Each stakeholder of a development team has to select quality measurements mostly without guidance rather than using a preselection based on choices made by stakeholders with a similar role.

In our research, we address the quantitative level of source code-related quality metrics, development tool-usage and repository-usage. Which is different to existing approaches, such as Goal Question Metric (GQM) or ISO/IEC standards on quality. These approaches try to model the quality of software through processes on multiple levels (e.g. management or customer level). The focus of our research is on tool usage-based **stakeholder's information needs** and the **tailoring of information** for SQA. *Individual information needs* of stakeholders are influenced by the role and the daily work in a software project. Our research will determine and describe the individual information needs of stakeholders, based on the type of software project (e.g. rich client, web application, etc.) and the development activities (e.g. bug fix, etc.). The information will be distilled from different repositories into a model that we call *SQA-Context*. This model represents a snapshot of the project based on tool-usage data of stakeholders (*stakeholder context*) and on information about the source code of a project (*technical context*). The key aspects of the stakeholder context are the actual roles of a stakeholder whereas the key aspects of the technical context are quality measurements such as code metrics. Once an SQA-Context is computed it can be used to tailor information according to the individual information needs.

## II. Motivating Example

The following example illustrates software development with a Web-based continuous integration platform, such as Jenkins (www.jenkins-ci.org), and a Web-based software quality platform for automatic source code analysis, such as Sonar (www.sonarsource.org). We consider software architects, developers and testers because of their essential role in the development process. We envision a generic approach which supports additional roles, which we may add at a later stage.

**Ann the Architect:** Ann uses standard modeling solutions for blueprints and Sonar to detect deviations from the blueprints in implemented software systems. The localization of deviations is done by an examination of different software quality measurements (e.g. coupling, cohesion, etc.). The variety of examined measurements heavily depends on the individual architect and project type (e.g. rich client, service, etc.). Ann uses a list of quality measurements to analyze each type of project. Such lists are shared between multiple architects, especially among new team members.

For each software project, Ann would need a list of quality measurements to facilitate the identification of potential violations of the architecture.

**Dave the Developer:** Dave is one out of two senior developers in his group and has contributed to almost every

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Doctoral Symposium

project of his team. He often has to interrupt his work on a project to solve a problem or at least support his colleagues to analyze the cause of a problem in another project. His experience from past projects is not the only reason for his analytical skills, he analyzes projects and problems based on a well structured approach. It is a combination of checks of software quality measurements in Sonar, source code changes in VCS, and build history in Jenkins. For example, he has a closer look on source code history if there is a series of build failures or a decrease in quality.

Dave would need a combined view onto selected software quality measurements, source code history, and build history based on the evolution of a software project to support the understanding of a problem cause.

**Tim the Tester:** Most of Tim's colleagues have a strong focus on testing those parts of the source code which are directly influenced by a change, for example a bug fix. This strong focus neglects all the other parts of the tested system, which maybe affected by the change or even unidentified bugs. Besides the usual testing, Tim always has a look into the source code history of a project and the evolution over time in Jenkins and Sonar. These lookups to different platforms and measurements slow down the work progress of Tim.

Tim would need a list of software quality measurements and events happening in VCS, Jenkins, and Sonar during and after changes in the source code of a software system.

## III. Research Idea and Proposed Solution

We envision an approach to bridge the gap between generic quality measurements and individual information needs of stakeholders (in our example Ann, Dave and Tim). Our solution should provide a context-sensitive tailoring of software quality measurements for an individual stakeholder in a software project based on an analysis of her tool-usage. Examples for tool-usage are inspected quality measurements in Web-based development tools, such as Jenkins or Sonar, and activities in repositories, such as VCS or bug trackers. We want to analyze tool-usage of stakeholders on artifacts in development tools and repositories with techniques known from Web-analytics [1]. With this information about tool-usage combined with needs from a qualitative study we want to find similarities within stakeholder groups from which we derive information needs of individual stakeholders for a context-sensitive tailoring. We use the notion of *context-sensitive* [13], which states: *"relation between the data, the world the data refers to, and the observer's expectations, intentions and interests"*.

Our understanding of a context-sensitive tailoring is a technique to automatically extract and present the essence of information in a meaningful way for each stakeholder.

Our main research question is: How can we better support different stakeholders in the quality assessment of a software system? Based on this we define following sub-questions:

1) How can information needs of stakeholders be described by employing qualitative and usage data of development tools and source code repositories?
2) How can such information needs be used to categorize stakeholder roles?
3) How can information tailoring based on tool-usage support propagation of changes in quality to stakeholders in a fast and accurate way?

### A. Approach

We tackle these three research questions with the following evolutionary approach:

*1) Stakeholder Context and Technical Context:* Stakeholder context and technical context describe a software project from different angles. A stakeholder context contains information about the organization in a project (e.g. roles) whereas the technical context contains information about the source code and specifications (e.g. requirements). We will model these contexts with an ontology, such as OWL, and a triplestore, such as RDF, to allow for an effective querying of relations with SPARQL and the possibility to shape implicit knowledge through reasoning.

**Initial investigation and consolidation:** We start with a comparison of related work in the field of modeling and describing contexts in software projects. In case of the stakeholder context we will analyze studies on information needs, Web-analytics and repository mining. As for the technical context, we will analyze studies on software evolution, software quality metrics and mining software repositories (MSR). The output will be a consolidated list of attributes.

**Attribute selection:** The outcome of the previous step allows one to select and assess attributes corresponding to different types of software projects. The attribute selection and assessment takes place based on standardized methods (e.g. balanced score board) and data from a software development company. The result is a list of attributes associated to a context.

**Heuristic context merging:** We will use a heuristic approach to merge the collected attributes into the specific context description (stakeholder and technical context). The merging of the stakeholder context is independent from the merging within the technical context. In the stakeholder context we primarily work with relative values (e.g. share of effort between stakeholders). For the technical context, we will normalize the different measurements before we merge them into the a common context description. The overall merging of both contexts into the final SQA-Context is described next. The results are heuristics to combine attributes in each context.

*2) Heuristic for SQA-Context Determination:* The SQA-Context is a core element of our approach and needs to describe the current status or situation in a software project without omitting essential information.

**Heuristic and model:** The SQA-Context provides a model for the merged result of the independent stakeholder and technical contexts. This model has to provide a consolidated overall view and is different from the two initial views, which have a focused view onto a part of the overall context. The output is a description of the SQA-Context model.

**Model dimensions:** Our approach will describe a software project based on three dimensions: (1) individual information

needs, (2) information needs based on the role of a stakeholder, and (3) quality measurements based on artifacts and source code. It is possible to add additional dimensions e.g. effort estimation, but we put the focus on the analysis of facts and not on effort estimation. The output is a description of the model dimensions.

**Determination of the SQA-Context:** The SQA-Context provides a technical view (e.g. without any individual information needs) and multiple individual views (e.g. with individual information needs) onto a software project. The heuristic context determination process takes each of the above three model dimensions into account and uses a weighting influenced by the frequency of changes on a dimension. The result is an SQA-Context filled with data.

*3) Information Tailoring based on SQA-Context:* The challenge for the information tailoring lies in the simplification to preserve expressivity. We claim that expressivity in a software quality assessment is mainly influenced by the purpose and the context of a information representation. Therefore, a context-based information tailoring must obtain essential information and has to provide a transition to less important or even background information. Next, we describe how to tailor information to achieve a high level of expressivity for software quality assessments.

**Tailoring mechanism:** Tailoring of information can be done either on measurement level (e.g. quality metric selection) or with a combination of measurement and structural criteria, for example, granularity (e.g. size of class). We tailor the information based on the granularity (e.g. class level, method level), frequency (e.g. daily, weekly) and measurements on the one hand and the individual information needs on the other hand. The result is a list of structural criteria.

**Architecture:** The output of the tailoring mechanism can be used for a visual representation but also for further processing of the data. We want to implement our approach as a prototypical Web service to provide a platform independent interface.

**Visualization:** Our primary idea is to support individuals during their daily work in a software project and foster their understanding of their view onto software quality. The visualization of tailored information is an essential part of the Web-based front-end, which we will devise. We want to implement a prototypical interface for mobile devices and workstations to visualize the tailored information.

### *B. SQA applied to the Illustrative Example*

Given our illustrative example, we support the three roles on different levels in a software project.

**Ann the Architect:** Our approach should support Ann and her team in tracking architectural changes based on information from different repositories. Ann will be provided with, for example, software quality attributes on package-level and their dependencies to specification documents (e.g. architecture map). Or software quality attributes with a focus on coupling between source packages will be presented in case of a change in the architecture map. With this, we can offer Ann a direct view onto changes in the software and facilitate the identification of potential violations of the architecture.

**Dave the Developer:** Our approach shall support Dave and his colleagues in filtering and highlighting issue and specification related information to their current work. This information contains the most recent software quality metrics on the code level (e.g. degree of coupling) and notifications on changes of artifacts (e.g. technical specification) related to a developer's work. During the identification of a problem cause, each developer should have a similar view as previous developers had on the software. This supports the understanding and should lead to better code quality.

**Tim the Tester:** Our approach shall support Tim and the other testers with information about events (e.g. a series of broken builds) occurring in different development tools during the implementation of, for example, a bug fix. This should raise the awareness of possible impacts or side effects caused by a change in the source code even in unexpected places. Based on this information it is possible to implement more precise test cases, which can lead to less bugs in a software system. A tailored view onto essential information can expedite the work of Tim and facilitate a comprehensive testing.

### *C. Research Plan*

We hypothesize that usage data analysis of different stakeholders in development tools and repositories can be used for a context-based tailoring of information to facilitate an SQA. Our hypothesis is evaluated by milestones:

**SQA-Context model:** The first step is to devise models describing the different contexts of our proposed solution. We then define methods and heuristics to populate the models in a way that they can provide a representative description for the current status of a project and allow further automatic processing. The goal of this milestone is a context description model that is processed and populated automatically based on different types of information repositories. We will analyze open source projects and evaluate the generated context description against the actual development in the project.

**Tailoring based on SQA-Context:** For the second milestone, we implement the tailoring mechanism of our approach. This tailoring mechanism decides what parts of the information should be shown to a stakeholder. The goal of this milestone is a mechanism to simplify the view onto complex information and allow a navigation without getting complex or loss of valuable information. We want to evaluate our tailoring mechanism against best practices through a case study in a software development company.

**Presentation & Interaction:** The final milestone implements the interaction with stakeholders and the presentation of the tailored information. The challenge lies in presenting the information in a way that is easy understandable for a stakeholder. During a software project, we want to evaluate our visualization (based on tailored information) in comparison to an existing software quality platform, such as Sonar.

### D. Validation

The validation will be done according to well-established methodologies [6] with focus on (1) individual information needs, (2) implementation of the platform, and (3) performance and usefulness of the overall approach. A study with architects, developers, and testers should provide deeper insights into their work and information needs. We will use the findings to define the SQA-Context and the according ontologies. The implemented platform will be used in a case study to validate the used measurements and the tailored information outcome. We will evaluate if the matching measurements with stakeholder information needs is adequate to the simulated situation (e.g. architectural change) during the development of a software system. Finally, a user study with a predefined set of tasks will be used to compare the productivity in a software project with our approach and a non-contextual SQA solution, such as Sonar. We will evaluate the productivity during the development of a software in terms of (1) platform usage (frequency and duration of visits), (2) time spent on an SQA, and (3) number of found bugs.

## IV. Related Work

Next, we provide a short overview of closely related research areas.

**Usage analysis & Information needs:** The research on the *information scent to model user information needs* [1] of Websites can be used as a basic input for our research to analyze the usage of development tools. Based on several studies, Sillito et al. [10] provide developer centered questions, which can be used for an initial description of information needs. The work of Ko et al. [7] on stakeholder roles in software projects can be used for a first categorization of stakeholder's information needs.
We want to combine these approaches to describe information needs of a stakeholder in a software project.

**Heuristics:** The mining of code repositories enables a wide range of possibilities to analyze code, and interactions between stakeholders. The direction of Hattori and Lanza tries to elicit the nature of commits in code repository [5]. The second direction in the field is contribution measurement is from Gousios et al. [4]. They extend the classic code focused measurement of developer contribution with artifacts, such as specification. Hattori and Gousios use heuristics to determinate the nature of a stakeholder's contribution.
For our approach we have to consolidate and expand existing heuristics and to create new ones if needed.

**Visualization of software quality metrics:** A survey on software visualization of Koschke *revealed a tendency to actually extend software visualization* [8] for a better understanding of measurements such as software quality metrics. Based on these tendency, Wettel and Lanza provide a visual way to analyze software quality metrics for the localization of software design disharmonies [12]. We want provide a visualization of automatically selected software metrics according to stakeholders of a software project.

## V. Conclusion and Outlook

Our main research question is: How can we better support different stakeholders in the quality assessment of a software system? For that we provided an illustrative example to realize our vision of the daily life of stakeholders in software projects. In our work, we will devise stakeholder context, technical context, heuristic context-determination, and context-based information tailoring to make our vision real. In particular, we will investigate the following approach in our research:

1) We will describe information needs of stakeholders by employing qualitative and usage data of development tools and source code repositories.
2) Based on such information needs we will categorize stakeholder roles.
3) We will tailor and propagate information about changes in quality to stakeholders in a fast and accurate way with respect to individual information needs and roles.

Our next steps are the definition of contexts (stakeholder and technical) and a survey on information needs in a software development team combined with a first tool-usage analysis of stakeholders with techniques known from usual Websites.

## References


[1] E. H. Chi, P. Pirolli, K. Chen, and J. Pitkow. Using information scent to model user information needs and actions and the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 490–497, NY, USA, 2001.

[2] S. R. Chidamber and C. F. Kemerer. Towards a metrics suite for object oriented design. In *Conference proceedings on Object-oriented programming systems, languages, and applications*, pages 197–211, NY, USA, 1991.

[3] P. B. Crosby. *Quality is free: the art of making quality certain*. New York: McGraw-Hill, 1979.

[4] G. Gousios, E. Kalliamvakou, and D. Spinellis. Measuring developer contribution from software repository data. In *Proceedings of the International Working Conference on Mining Software Repositories*, pages 129–132, NY, USA, 2008.

[5] L. Hattori and M. Lanza. On the nature of commits. In *Automated Software Engineering - Workshops*, pages 63–71, 2008.

[6] B. Kitchenham. A methodology for evaluating software engineering methods and tools. In *Experimental Software Engineering Issues*, pages 121–124, 1992.

[7] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of the International Conference on Software Engineering*, pages 344–353, Washington, USA, 2007.

[8] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *Journal of Software Maintenance*, 15(2):87–109, Mar. 2003.

[9] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering*, pages 521–530, NY, USA, 2008.

[10] J. Sillito, G. C. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 23–34, NY, USA, 2006.

[11] S. A. Slaughter, D. E. Harter, and M. S. Krishnan. Evaluating the cost of software quality. *Commun. ACM*, 41(8):67–73, Aug. 1998.

[12] R. Wettel and M. Lanza. Visually localizing design problems with disharmony maps. In *Proceedings of the 4th ACM symposium on Software visualization*, pages 155–164, NY, USA, 2008.

[13] D. D. Woods, E. S. Patterson, and E. M. Roth. Can we ever escape from data overload? a cognitive systems diagnosis. *Cognition, Technology & Work*, 4(1):22–36, 2002.